This program, scheduler.py, allows you to see how different schedulers perform
under scheduling metrics such as response time, turnaround time, and total
wait time. Three schedulers are "implemented": FIFO, SJF, and RR.

There are two steps to running the program.

First, run without the -c flag: this shows you what problem to solve without
revealing the answers. For example, if you want to compute response,
turnaround, and wait for three jobs using the FIFO policy, run this:

```
  ./scheduler.py -p FIFO -j 3 -s 100
```

If that doesn't work, try this:

```
  python ./scheduler.py -p FIFO -j 3 -s 100
```

This specifies the FIFO policy with three jobs, and, importantly, a specific
random seed of 100. If you want to see the solution for this exact problem,
you have to specify this exact same random seed again. Let's run it and see
what happens. This is what you should see:

```
prompt> ./scheduler.py -p FIFO -j 3 -s 100
ARG policy FIFO
ARG jobs 3
ARG maxlen 10
ARG seed 100

Here is the job list, with the run time of each job:
  Job 0 (length = 1)
  Job 1 (length = 4)
  Job 2 (length = 7)
```

Compute the turnaround time, response time, and wait time for each job.
When you are done, run this program again, with the same arguments,
but with -c, which will thus provide you with the answers. You can use
to use -s <somenumber> or your own job list (-l 10,15,20 for example)
to generate different problems for yourself.

As you can see from this example, three jobs are generated: job 0 of length 1,
job 1 of length 4, and job 2 of length 7. As the program states, you can now
use this to compute some statistics and see if you have a grip on the

basic
concepts.

Once you are done, you can use the same program to "solve" the problem and see
if you did your work correctly. To do so, use the "-c" flag. The output:

```
prompt> ./scheduler.py -p FIFO -j 3 -s 100 -c
ARG policy FIFO
ARG jobs 3
ARG maxlen 10
ARG seed 100

Here is the job list, with the run time of each job:
  Job 0 (length = 1)
  Job 1 (length = 4)
  Job 2 (length = 7)

** Solutions **

Execution trace:
  [time   0] Run job 0 for 1.00 secs (DONE)
  [time   1] Run job 1 for 4.00 secs (DONE)
  [time   5] Run job 2 for 7.00 secs (DONE)

Final statistics:
  Job   0 -- Response: 0.00  Turnaround 1.00  Wait 0.00
  Job   1 -- Response: 1.00  Turnaround 5.00  Wait 1.00
  Job   2 -- Response: 5.00  Turnaround 12.00  Wait 5.00

  Average -- Response: 2.00  Turnaround 6.00  Wait 2.00
```

As you can see from the figure, the -c flag shows you what happened. Job 0 ran
first for 1 second, Job 1 ran second for 4, and then Job 2 ran for 7 seconds. Not too hard; it is FIFO, after all! The execution trace shows these
results.

The final statistics are useful too: they compute the "response time" (the
time a job spends waiting after arrival before first running), the "turnaround
time" (the time it took to complete the job since first arrival), and the
total "wait time" (any time spent ready but not running). The stats are shown
per job and then as an average across all jobs. Of course, you should have

computed these things all before running with the "-c" flag!

If you want to try the same type of problem but with different inputs, try
changing the number of jobs or the random seed or both. Different random seeds
basically give you a way to generate an infinite number of different problems
for yourself, and the "-c" flag lets you check your own work. Keep doing this
until you feel like you really understand the concepts.

One other useful flag is "-l" (that's a lower-case L), which lets you specify
the exact jobs you wish to see scheduled. For example, if you want to find out
how SJF would perform with three jobs of lengths 5, 10, and 15, you can run:

```
prompt> ./scheduler.py -p SJF -l 5,10,15
ARG policy SJF
ARG jlist 5,10,15

Here is the job list, with the run time of each job:
  Job 0 (length = 5.0)
  Job 1 (length = 10.0)
  Job 2 (length = 15.0)
...
```

And then you can use -c to solve it again. Note that when you specify the
exact jobs, there is no need to specify a random seed or the number of jobs:
the jobs lengths are taken from your comma-separated list.

Of course, more interesting things happen when you use SJF (shortest-job
first) or even RR (round robin) schedulers. Try them and see!

And you can always run

```
  ./scheduler.py -h
```

to get a complete list of flags and options (including options such as setting
the time quantum for the RR scheduler).